FAKULTA
INFORMAČNÍCH
TECHNOLOGIÍ
ČVUT V PRAZE

ZADÁNÍ BAKALÁŘSKÉ PRÁCE

Název:	Spolehlivá záloha dat na optická média
Student:	František Pivovarník
Vedoucí:	Ing. Viktor Černý
Studijní program:	Informatika
Studijní obor:	Informační technologie
Katedra:	Katedra počítačových systémů
Platnost zadání:	Do konce zimního semestru 2019/20

Pokyny pro vypracování

1. Analyzujte problematiku zálohy dat na optická média. Zaměře se na tyto faktory: bezpečnost uložení dat (redundance), cena a datová kapacita.
2. Analyzujte, jakými způsoby lze zrekonstruovat data z více identických, ale různě poškozených optických disků.
3. Na základě analýzy vyhodnoťte nejlepší metody zálohy dat na optická média ze všech tří různých kritérií.
4. Nejvýhodnější metody, které zvolí vedoucí práce, implementujte v operačním systému Linux.
5. Výsledkem práce bude zpracovaná problematika zálohy dat na optické média a softwarové řešení takové zálohy.

Seznam odborné literatury

Dodá vedoucí práce.

prof. Ing. Pavel Tvrdík, CSc.
vedoucí katedry

doc. RNDr. Ing. Marcel Jiřina, Ph.D.
děkan

V Praze dne 13. března 2018

FAKULTA
INFORMAČNÍCH
TECHNOLOGIÍ
ČVUT V PRAZE

Bakalárska práca

Spoľahlivá záloha dat na optické média

František Pivovarník

Katedra počítačových systémů
Vedúci práce: Ing. Viktor Černý

15. mája 2018

Prehlásenie

Prehlasujem, že som predloženú prácu vypracoval(a) samostatne a že som uviedol(uviedla) všetky informačné zdroje v súlade s Metodickým pokynom o etickej príprave vysokoškolských záverečných prác.

Beriem na vedomie, že sa na moju prácu vzťahujú práva a povinnosti vyplývajúce zo zákona č. 121/2000 Sb., autorského zákona, v znení neskorších predpisov, a skutočnosť, že České vysoké učení technické v Praze má právo na uzavrenie licenčnej zmluvy o použití tejto práce ako školského diela podľa § 60 odst. 1 autorského zákona.

V Prahe 15. mája 2018 .....

Odkaz na túto prácu

Pivovarník, František. *Spoľahlivá záloha dat na optické média.* Bakalárska práca. Praha: České vysoké učení technické v Praze, Fakulta informačních technologií, 2018.

Abstrakt

Táto práca sa zaoberá problematikou zálohovania dát na optické média a obnovou dát z poškodených optických diskov. Popisuje rôzne metódy zálohovania a typy optických diskov. Ďalej rieši problém obnovy dát z dvoch rovnakých, ale rôzne poškodených optických diskov. Tento problém riešime implementáciou knižnice jazyka C a jednoduchého nátroja, ktorý pracuje s knižnicou.

Klíčová slova knižnica, zálohovanie, obnova dát, optické média, Linux, C

Abstract

This thesis focuses on data backup to optical media and data restoration from corrupted optical media. It describes various methods of data backup and various types of optical discs. It also resolves problem of data restoration from two identical, but differently corrupted optical discs. This problem is solved by implementation C library and small utility which works with this library.

Keywords library, backup, data restoration, optical media, Linux, C

Obsah

Zoznam tabuliek

Úvod

Zálohovanie dát zohráva dôležitú úlohu pri ochrane našich dát pred poškodením alebo stratou. Dáta sa zálohujú, aby sme predišli ich strate pri zlyhaní hardvéru, mechanickom poškodení hardvéru alebo pri neúmyselnom zmazaní dát. Typy zálohovaní a média na ktoré dáta zálohujeme sa samozrejme vyvíjajú a zlepšujú. Cieľom je bezpečná záloha dát, pričom chceme čo najvyššiu kapacitu média za čo najnižšiu cenu.

V dnešnej dobe sú medzi bežnými užívateľmi na zálohovanie dát používané najmä klasické alebo externé pevné disky. Avšak stále sa používajú aj optické média. Tie boli rozšírenejšie v minulosti a mnoho užívateľov má preto svoje dáta z minulosti stále zálohované na optických médiách.

Optické média tak ako všetko starnú a ich životnosť nie je neobmedzená. Stáva sa, že vplyvom stranutia alebo mechanického poškodenia sa optické disky stávajú nečitateľnými. V takých prípadoch užívateľ stráca časť svojich dát, v najhoršom prípade aj všetky svoje zálohované dáta. V tejto práci sa preto zaoberáme obnovou dát z dvoch rozdielne poškodených optických diskov.

Cieľom práce je preto oboznámenie sa so zálohovaním na optické média, analýza jednotlivých metód zálohovania, vytvorenie knižnice a nástroja na obnovu dát z poškodených optických diskov. Pri zálohovaní dát je využitý princíp redundancie, to využívame aj pri obnovovaní z poškodených diskov.

Štruktúra

V prvej časti práce sa zaoberáme analýzou problematiky zálohovania dát na optické média. Rozoberáme rôzne metódy zálohovania dát, typy optických diskov na ktoré dáta zálohujeme a ich vlastnosti. V tejto časti ešte popisujeme výhody zálohovania na optické média, analyzujeme funkcionalitu a existujúce riešenia na obnovu dát z poškodených diskov.

Druhá časť patrí návrhu nášho riešenia. V nej sa venujeme popisu jednotlivých kľučových častí nášho návrhu obnovy dát. Zároveň popisujeme ako jednotlivé časti implementujeme.

Tretia časť s názvom realizácia sa zaoberá implementáciou nášho riešenia. Popisujeme knižnicu použitú na prácu so súbormi a jej funkcie použité v našej implementácii. Ďalej detailne popisujeme našu implementáciu, všetky funkcie a štruktúry. Ich funkcionalitu a priebeh.

Na záver sa venujeme popisu testovania našej implementácie na fyzických zariadeniach.

Kapitola 1

Analýza

1.1 Zálohovanie dát

Zálohovanie dát znamená kopírovanie súborov alebo databáz na sekundárne miesto na uchovanie v prípade výpadku zariadenia alebo inej chyby. Proces zálohovania údajov je kľúčový pre úspešnú obnovu dát po havarii.[1]

1.1.1 Základné typy záloh

Zálohy sa dajú rozdeliť podľa [2] na nasledujúce typy:

1. Úplná záloha
2. Inkrementačná záloha
3. Rozdielová záloha
4. Zrkadlová záloha

1.1.2 Úplná záloha

Je to metóda zálohovania pri ktorej všetky súbory a priečinky vybrané na zálohovanie budú zálohované. Zvyčajne sa využíva ako počiatočné alebo prvé zálohovanie, po ktorom ďalej nasleduje inkrementačné alebo rozdielové zálohovanie. Po niekoľkých inkrementačných alebo rozdielových zálohovaniach je bežné začať znova čerstvým úplným zálohovaním. Používanie len úplnej zálohy sa typicky hodí pri zálohovaní menších dát.
Výhody:

- Obnova je rýchlá a jednoduchá
- Jednoduché získať a obnoviť rôzne verzie

Nevýhody:

- Zálohovanie môže trvať veľmi dlho, keďže každý súbor je zálohovaný pri každom spustení úplného zálohovania
- Zaberá najviac úložného miesta v porovnaní s ostatnými metódami. Opakované ukladanie rovnakých súborov má za výsledok neefektívne využívanie úložného priestoru

1.1.3 Inkrementačná záloha

Ide o zálohovanie všetkých zmien vykonaných od posledného zálohovania. Posledná záloha môže byť úplná alebo jednoducho posledná inkrementačná záloha. Pri použití tohto typu zálohy je najprv použitá úplná záloha a nasledujúce zálohy sú len zmeny v súboroch a pridávanie novo vytvorených súborov od poslednej zálohy.
Výhody:

- Rýchlejšie zálohovanie
- Efektívne využívanie úložného miesta, keďže súbory nie sú duplikované
- Využíva omnoho menej miesta v porovaní s vykonávaním len úplných záloh, tiež je úspornejšia oproti rozdielovej zálohe

Nevýhody:

- Obnova dát je pomalšia ako pri úplnej a rozdielovej zálohe
- Obnova dát je komplikovanejšia. Všetky zálohy (najprv prvá úplná záloha a potom všetky nasledujúce inkrementačné zálohy) sa musia postupne obnoviť

1.1.4 Rozdielová záloha

Tento typ zálohy spadá niekde medzi úplnú a inkrementačnú zálohu. Táto záloha vykonáva zálohu všetkých zmien vykonaných od poslednej úplnej zálohy. Táto metóda vyžaduje počiatočnú úplnú zálohu, každá ďalšia záloha ukladá zmeny vykonané od tejto úplnej zálohy. Vo výsledku sa jedná o rýchlejšiu metódu ako je úplná záloha. Úložného priestoru využíva menej ako úplná, ale viac ako inkrementačná záloha. Obnova dát je pomalšia ako pri obnovovaní úplnej zálohy, ale je rýchlejšia ako obnova inkrementačnej zálohy.
Výhody:

- Rýchlejšia ako úplná záloha
- Efektívnejšie využívanie miesta vďaka zálohovaniu zmien od poslednej úplnej zálohy

- Rýchlejšia obnova dát ako pri inkrementačnej zálohe

Nevýhody:

- Pomalšia ako inkrementačná záloha
- Menej efektívna pri využívaní úložného priestoru ako inkrementačná metóda
- Pomalšia obnova dát ako pri úplnej zálohe
- Obnova dát je komplikovanejšia ako pri úplnej, ale jednoduchšia ako pri inkrementačnej zálohe. Obnovuje sa počiatočná úplná záloha a posledná vykonaná rozdielová záloha

1.1.5 Zrkadlová záloha

Nie všetci uznávajú túto metódu za zálohovanie. Ako názov naznačuje jedná sa o zálohu zrkadleného zdroja dát. Pri tomto type zálohy, keď je zmazaný súbor v zdroji, tak je tento súbor nakoniec zmazaný aj zo zrkadlovej zálohy. Kvôli tejto vlastnosti musí byť zrkaldové zálohovanie dát používané opatrne. Ak bol súbor v zdrojových dátach vymazaný omylom, išlo o sabotáž, alebo bol vymazaný vírusom, tak sa to môže prejaviť na zálohovaných dátach, kde sa tento súbor taktiso zmaže. Tým pádom o tento súbor prídeme.
Výhody:

- Záloha je čistá a neobsahuje staré a neaktuálne súbory

Nevýhody:

- Existuje šanca, že súbory vymazané zo zdroja nevedomky alebo nechtiac budú zmazané aj zo zrkadlovej zálohy

1.2 Optický disk

Optický disk je úložné zariadenie s priamym prístupom, na ktorý sú dáta zapisované a sú z neho čítané použitím laserového svetla. Najbežnejšie typy sú CD, DVD a Blu-ray disky.

Ako vymeniteľné média optické disky nahradili predtým používané magnetické disky. Optické médiá majú nižšiu váhu, vyššiu kapacitu a sú menej náchylné na poškodenie zapríčinené extrémnymi teplotami.

1.2.1 CD (Compact Disk)

Optické médium kruhového tvaru s priemerom 120 mm a hrúbkou 1,2 mm. Váha sa pohybuje od 15 do 20 gramov. Kapacita je 80 minút zvukového záznamu, alebo 700 MB dát.

CD sú vyrobené z polykarbonátu. Povrch disku je obalený tenkou hliníkovou vrstvou, vďaka ktorej je povrch reflexný. Všetko je to obalené vrstvou laku, ktorá slúži ako ochranná vrstva.

Dáta sú ukladané ako za sebou nasledujúce malé voľným okom neviditeľné zárezy usporiadané v špirálovitej stope. Vzdialenosť medzi týmito stopami je 1,6 μm.

Na čítanie z CD sa používa laser s vlnovou dĺžkou 780 nm. Rozdiely vo výške medzi zárezmi a rovnou plochou na CD majú za výsledok odraz svetla v rozdielnej intenzite.[3]
Formáty CD[4]:

- **CD-ROM** Compact Disc Read Only Memory je disk len na čítanie. Disky tohto typu sú vytvárané na komerčné účely. Keď sa raz na disk niečo zapíše, neskôr sa sa dáta na tento disk ukladať nedajú.
- **CD-R** Compact Disc Recordable je disk typu WORM (Write Once Read Multiple) zapíš raz, čítaj viackrát. Na tieto disky môžu byť dáta zapísane len raz. Potom ostanú na disku permanentne. Čítané z neho môžu byť ako zo štandardného CD-ROM. Po zapísaní dát sa z CD-R stáva CD-ROM.
- **CD-RW** Compact Disc Re-Writable je to zmazateľný disk. Dáta môžu byť z tohto disku zmazané a nahrané späť mnohokrát.

1.2.2 DVD (Digital Versatile Disc)

Optické médium kruhového tvaru, na prvý pohľad vyzerá rovnako ako CD. Má rovnaký priemer 120 mm, jeho hrúbka sa pohybuje od 0,6 mm do 1,2 mm. Rovnako ako CD sa aj DVD vyrábajú z polykarbonátu, majú reflexnú hliníkovú vrstvu a ochrannú vrstvu.

Existuje viacero druhov DVD, jedno alebo obojstranné s jednou alebo dvomi vrstvami. Preto sa kapacita DVD pohybuje v rozmedzí od 4,7 GB(jednostranné DVD s jednou vrstvou) až po 17 GB(obojstranné DVD s dvomi vrstvami).

Dáta sú ukladané rovnako ako na CD, teda ako malé zárezy v špirálovej stope. Jednotlivé stopy sú bližšie k sebe, to umožňuje mať viac stôp na disku s rovnakými rozmermi. Miesto medzi stopami sa oproti CD znížilo na 0,74 μm.[5][6]

Formáty DVD[4]:

- **DVD-R** Je to zapisovateľný DVD formát podobne ako DVD+R alebo CD-R. Dáta na disk môžeme nahrať len raz, potom ostávajú na disku permanentne.
- **DVD+R** To isté ako DVD-R. Zápis dát len raz, potom sa dá už len čítať.
- **DVD-RW** Ide o zmazateľný formát podobne ako DVD+RW alebo CD-RW. Dáta sa dajú z disku opakovane vymazovať a zapisovať.
- **DVD+RW** Rovanký formát ako DVD-RW.

1.2.3 Blu-ray

Názov Blu-ray Disc je odvodený od modrého (anglicky blue) laseru používaného na čítanie a zápis tohto typu diskov.

Ako pri predošlých dvoch typoch optických diskov aj tentokrát sa jená o médium kruhového tvaru s priemerom 120 mm a hrúbkou 1,2 mm. Opäť sa jedná o médium vyrábané z polykarbonátu a samozrejme obsahuje aj reflexnú vrstvu.

Blu-ray disk obsahuje 1-4 vrstvy z ktoých má každá vrstva kapacitu 25 GB. Teda maximálnu 100 GB kapacitu má štvorvrstvový Blu-ray disk.

Ukladanie dát je postavené na rovnakom princípe ako pri CD a DVD. To znamená malé zárezy v špirálovej stope ktorá ide od stredu disku smerom k jeho okrajom. Stopy sú k sebe ešte bližšie, medzery medzi nimi sú 0,32 μm. Vlnová dĺžka sa znížila na 405 nm, to umožnuje čítanie informácii z menších zárezov. Vďaka tomu sa zvýšila kapacita na spomínaných 25GB.[7]
Formáty Blu-ray:

- **BD-RE** Blu-ray Disc Rewritable je prepisovateľný disk podobne ako DVD-RW a CD-RW. Umožňuje dáta z disku vymazať a zapísať nové.
- **BD-R** Blu-ray Disc Recordable je disk typu WORM, to znamená, že zápis dát je možný len raz. Potom môžeme dáta z disku len čítať.
- **BD-ROM** Blu-ray Disc Read Only Memory je disk na ktorom sú nahrané dáta a užívateľ je schopný len čítania. Obsah disku je chránený pred nelegálnym kopírovaním pomocou troch vrstiev správy digitálnych práv.

1.3 Záloha dát na optické média

Optické média už nie su tak populárnou voľbou na zálohu dát ako v minulosti. Je to hlavne vďaka stále klesajúcej a nízkej cene za klasické mechanické pevné

disky, ktoré majú oveľa vyššiu kapacitu a cena za 1 GB úložného priestoru sa pohybuje na rovnakej alebo nižšej úrovni ako cena pri optických médiách. Napriek tomu majú optické disky stále niekoľko výhod.

- Nízka cena pri zálohe malého množstva dát. V dnešnej dobe sa bežne predávajú pevné disky s kapacitou minimálne 500 GB. Ak však potrebujeme zálohovať len malé množstvo dát je menej finančne náročné zálohovať tieto dáta na niekoľko optických diskov ako na jeden pevný disk.

- Dlhá životnosť optických diskov je ďalšou výhodou oproti mechanickým pevným diskom. DVD disky majú podľa výrobcov životnosť 10 až 25 rokov. Blu-ray disky sú na tom ešte lepšie a vďaka ich technológii vydržia ešte dlhšie ako DVD. Tento odhad samozrejme neplatí ak sú disky vystavované extrémnym podmienkam, napríklad extrémne teploty, vysoká vlhkosť alebo silné priame svetelné žiarenie. Taktiež sa životnosť skracuje každodenným používaním a prenášaním. Tu sa zvyšuje riziko poškrabania alebo iného mechanického poškodenia. Avšak ak tieto disky používame na zálohu dát tak sa nepoužívajú často a sú uložené na bezpečnom mieste kde nie sú vystavené extrémnym vonkajším vplyvom.

Pre zvýšenie bezpečnosti našich dát zálohovaných na optických diskoch využívame redundanciu. To znamená, že rovnaké dáta máme uložené na dvoch alebo viacerých diskoch. Tým znižujeme pravdepodobnosť, že o naše dáta prídeme kvôli poškodeniu optického disku. Disky s rovnakými dátami je dobre uchovávať na rozdielnych miestach, čím zabránime prípadnému zničeniu oboch kópií napríklad pri požiari. Aj pri malom poškodení oboch diskov je pravdepodobnosť poškodenia rovnakých sektorov na oboch diskoch veľmi malá, teda stále sme schopný tieto dáta obnoviť použitím oboch diskov a nášho programu.

Rovnaké dáta môžeme ukladať na viacero diskov využitím rôznych spôsobov. Dajú sa využívať princípy, ktoré poznáme z ukladania dát v rôznych typoch diskových polí.

1.4 Popis požadovanej funkcionality

Hlavnou požiadavkou na funkčnosť programu je obnova dát z poškodených optických diskov. Konkrétne ide o obnovu z dvoch optických diskov, pričom na obidvoch týchto diskoch sa nachádzajú rovnaké dáta. Našou úlohou je nájsť poškodené sektory na oboch diskoch, určiť či náhodou nie sú poškodené na oboch diskoch rovnaké sektory, v tom prípade sa dáta nebudú dať obnoviť v plnom rozsahu. Ak sú poškodené sektory rozdielne na oboch diskoch, program dokáže tieto dáta obnoviť do pôvodnej funkčnej formy.

1.5 Produkty s podobnou funkcionalitou

Takýchto produktov existuje viacero, niektoré sú zamerané len na obnovu z pevných diskov, iné zvládajú aj obnovu z optických diskov, USB kľúčov, pamäťových kariet a iných zariadení. Tieto produkty dokážu obnoviť dáta stratené dôsledkom rôznych operácii alebo chýb[8].

- Obnova vymazaných súborov
- Obnova súborov stratených po formátovaní disku
- Obnova súborov z čiastočne chybného disku alebo disku po jeho zlyhaní

Obnova zmazaných súborov je možná vďaka tomu, že keď zmažeme súbor, operačný systém ho v skutočnosti z disku nevymaže. Operačný systém si ukladá informácie o uložených súboroch pomocou takzvaných ukazovateľov. Každý súbor a priečinok na pevnom disku má ukazovateľ, ktorý operačnému systému hovorí kde blok daných dát začína a kde končí.

Keď zmažeme súbor, operačný systém odstráni príslušný ukazovateľ a označí dané sektory na disku ako dostupné. Z pohľadu súborového systému preto nie je súbor naďalej na disku a sektory ktoré mu patrili sú považované za voľné miesto.

Avšak, až kým operačný systém nezapíše do týchto sektorov nové dáta, nami zmazaný súbor je stále obnoviteľný. Softvér na obnovu dát prehľadáva pevný disk a obnovuje všetky tieto zmazané súbory.

Po formátovaní disku alebo jeho partícii funguje obnova dát podobne ako pri klasickom vymazaní súborov.

Obnova dát z chybného alebo už takmer nefunkčného disku je náročnejšia ako obnova vymazaných dát. Preto nedosahuje takú úspešnosť ako obnova vymazaných dát. Samozrejme existujú produkty na obnovu dát z chybných diskov, ale najväčšia šanca na úspech je obrátiť sa na profesionálnu službu kde sa zaoberajú obnovou dát z chybných diskov.

Ak sú na disku vadné len niektoré sektory, stále môžeme z disku vytiahnuť dáta uložené v týchto funkčných sektorov a použiť softvér ktorý vie obnovovať niektoré typy súborov, napr. grafické súbory, videá, zvukové súbory, dokumenty a maily[9].

Nenašli sme však žiadny nástroj na obnovu dát, ktorý by vedel pracovať s dvoma optickými diskami, na ktorých sú uložené identické dáta. Ale tieto optické disky sa mechanickým vplyvom alebo vplyvom starnutia disku poškodili a teda nie je možné z nich čítať všetky sektory.

Kapitola 2

Návrh

2.1 Načítanie dát

Ak užívateľ má na svojom zariadení k dispozícii optickú mechaniku, tak väčšinou má práve jednu takúto mechaniku. Preto náš program pracuje s jednou optickou mechanikou. Pre úspešné načítanie dát je potrebné, aby mal užívateľ na svojom pevnom disku dostatok voľného miesta na nakopírovanie dvoch optických diskov.

Načitavanie dát prebieha po jednotlivých sektoroch veľkosti 2048 bajtov. Tieto sektory načítavame do bufferu, ktorého veľkosť môže užívateľ meniť. Štandardne je veľkosť bufferu nastavená na 10 000 sektorov, tj. 20 480 000 bajtov = 20,48 MB.

Ak narazíme na sektor, ktorý náš program nedokáže prečítať, preskočíme celý blok dát. Taktiež si uložíme informáciu o počiatočnom bajte tohto bloku. Blok dát ma veľkosť 500 sektorov. Do bufferu vložíme 500 sektorov, pričom každý bajt týchto sektorov je nastavený na hodnotu `NULL`. Ak po preskočení bloku dát načítame nasledujúci sektor úspešne pokračujeme načítavaním po sektoroch. Ak je aj ďalší sektor chybný, opäť preskakujeme ďalší blok dát rovnakej veľkosti. Toto opakujeme vždy pri neúspešnom pokuse o načítanie sektoru.

Preskočený blok dát nemusí obsahovať len chybné sektory. Niektoré sektory v tomto bloku môžu byť plne funkčné, avšak je málo pravdepodobné, že je poškodený len jeden sektor a nie viacero za sebou nasledujúcich. Blok preskakujeme hlavne kvôli časovej náročnosti pokusu o čítanie chybného sektora.

Jednotlivé sektory načítavame, až kým nenarazíme na koniec dát uložených na optickom disku.

2.2 Informácie o načítaných obrazoch diskov

Dáta potrebné k ďalšej práci s načítanými obrazmi diskov máme uložené v štruktúre. V nej máme uloženú cestu k danému súboru a jeho veľosť v bajtoch.

Ďalej máme v tejto štruktúre ukazovateľ na súbor s naším obrazom disku, tento ukazovateľ je dôležitý pri I/O operáciách v jazyku C.

Nakoniec máme pre každý obraz disku uložené informácie o chybných blokoch dát. Pozície týchto chybných blokov máme uložené v dynamickom poli. Aby sme mohli toto pole dynamicky zväčšovať, máme uloženú aj informáciu o aktuálnej veľkosti tohto poľa. Keďže veľkosť poľa môže byť rozdielna od počtu uložených chybných blokov, v našej štruktúre máme aj informáciu o počte prvkov v poli chybných blokov dát.

2.3 Buffer

Spomínaný buffer do ktorého načítavame sektory má prednastavenú veľkosť na 20,48 MB. Túto veľkosť však môže užívateľ meniť podľa svojho uváženia pri spustení danej úlohy v programe.

Vždy, keď je buffer plný načítaných sektorov z optického disku, vypíšeme všetky v ňom uložené dáta na pevný disk a skočíme na začiatok bufferu a pokračujeme v načítavaní. Takto postupne skopírujeme celý obsah optického disku na náš pevný disk.

2.4 Spojenie funkčných dát z dvoch optických diskov

Po úspešnom načítaní oboch optických diskov s našimi zálohovanými dátami máme na disku uložené ISO obrazy oboch diskov a v pamäti uložené informácie o chybných blokoch sektorov na každom z týchto diskov.

V tomto kroku ide o načítanie blokov sektorov označených ako chybné v obraze prvého disku z druhého disku. Postupne sa posúvame v poli, v ktorom sú uložené informácie o chybných blokoch na disku číslo jeden. Tieto bloky sa snažíme načítať z obrazu disku číslo dva a nahradiť ich na miestach, kde sú v prvom obraze na miesto chybných blokov sektorov dosadené hodnoty `NULL`.

Ak sú dané bloky na druhom disku v poriadku, to znamená že sa nenachádzajú v poli, kde máme uložené všetky chybné bloky na druhom disku, tak sa jednotlivé bajty postupne skopírujú a kombináciou nepoškodených bajtov z disku jeden a disku dva získame funkčný obraz disku v ktorom sú pôvodné dáta, ktoré boli nahrané na poškodených diskoch.

Druhá možnosť je, že narazíme na bloky sektorov ktoré sa prekrývajú na jednotlivých diskoch. Nemusia byť chybné bloky začínajúce na rovnakých miestach na jednotlivých diskoch. Stačí ak sa časti týchto blokov prekrývajú. V tomto prípade všetky sektory ktoré sú označené ako chybné na oboch diskoch skúsime načítať znova, tentokrát však postupne sektor po sektore, bez preskakovania celých blokov.

Ak aj po tejto operácii budeme mať rovnaké sektory chybné na oboch diskoch, môžeme prehlásiť, že dáta z týchto dvoch poškodených optických diskov nevieme úplne obnoviť.

2.5 Výstup programu

Po vykonaní všetkych vyššie popisovaných častí programu dospejeme buď k záveru že dáta nedokážeme obnoviť alebo práve naopak dáta obnovíme.

V prípade že dáat vieme obnoviť je výstupom programu obraz optického disku vo formáte ISO. Tento obraz obsahuje funkčné dáta a je pripravený na vypálenie na nový optický disk. Na vypaľovanie optických diskov existuje množstvo nástrojom, preto sme sa touto časťou nezaoberali.

Kapitola 3

Realizácia

3.1 Knižnica cstdio

V tejto sekcii čerpáme informácie z manuálových stránok knižnice cstdio, tie nájdeme na adrese [10]. Knižnica jazyka C vykonávajúca vstupné a výstupné operácie. Táto knižnica používa takzvané streamy. Tie slúžia na operácie s fyzickými zariadeniami akými sú klávesnice, tlačiarne, terminály alebo všelijaké iné typy súborov podporované systémom. Streamy sú abstraktom pre interakciu s týmito zariadeniami a súbormi jednotným spôsobom. Všetky streamy majú podobné vlastnosti nezávisle od individualných vlastností fyzického média s ktorým sú spojené.

V knižnici stdio sa so streamami zaobchádza ako s ukazovateľmi na objekty typu `FILE`. Ukazovateľ na objekt typu `FILE` jedinečne identifikuje stream a je používaný ako parameter pre operácie zahŕňajúce daný stream.

Existujú tri štandardné streamy, ktoré sú automaticky vytvorené a otvorené pre všetky programy používajúce túto knižnicu.

3.1.1 Vlastnosti streamov

Streamy majú vlastnosti, ktoré definujú aké funkcie môžu byť na streame použité a ako tieto vlastnosti ovplyvňujú zaobchádzanie s datovým vstupom alebo výstupom. Väčšina týchto vlastností je definovaná pri otvorení súboru priradenému k streamu.

3.1.1.1 Prístup na čítanie/zápis

Špecifikuje, či má daný stream prístup na čítanie alebo zápis, prípadne na oboje do fyzického média s ktorým je spojený.

3.1.1.2 Textový alebo binárny mód

Textové streamy reprezentujú skupinu riadkov textu, kde každý riadok končí znakom pre nový riadok. V závislosti na prostredí v ktorom aplikácia beží, sa môže stať, že niektoré znaky budú použitím streamu preložené kvôli ich špeciálnemu významu, ktorý je definovaný pre dané prostredie.

Na druhej strane binárne streamy sú sekvenciou znakov zapisovaných alebo čítaných z fyzického média bez prekladu. Teda sú totožné jeden znak k jednému so znakmi čítanými alebo zapisovanými do streamu.

3.1.1.3 Buffer

Buffer je blok v pamäti do ktorého sú dáta ukladané predtým ako sú fyzicky čítané alebo zapisované do príslušného súboru alebo zariadenia. Existujú úplne bufferované, riadkovo bufferované alebo nebufferované streamy. Pri úplne bufferovaných streamoch sú dáta zapisované alebo čítané, keď je buffer plný. Pri riadkovo bufferovaných streamoch sa zápis alebo čítanie vykoná, keď narazí na znak nového riadku. Znaky v nebufferovaných streamoch sú určené na zápis alebo čítanie čo najskôr ako je to možné.

3.1.2 Indikátory v streamoch

Špecifikujú momentálny stav streamu, ktorý ovplyvňuje správanie sa niektorých vstupných a výstupných operácií na streame.

3.1.2.1 Indikátor chyby

Tento indikátor je nastavený ak nastane chyba pri niektorej operácii na streame. Kontrolujeme ho funkciou `ferror` a indikátor sa resetuje použitím jednej z funkcií `cleanerr`, `freopen` alebo `rewind`.

3.1.2.2 Indikátor konca súboru

Ak je nastavený, znamená to, že posledná operácia čítania alebo zápisu vykonaná na danom streame dosiahla koniec súboru. Stav zisťujeme funkciou `feof` a resetujeme ho funkciami `cleanerr` alebo `freopen`. Resetuje sa taktiež volaním ktorejkoľvek funkcie na zmenu pozície v streame.

3.1.2.3 Indikátor pozície

Je to interný ukazovateľ v každom streame, ktorý ukazuje na znak, ktorý bude čítaný alebo zapísaný pri nasledujúcej operácií. Jeho hodnota je získavaná funkciami `ftell` a `fgetpos`. Pozíciu môžeme meniť funkciami `rewind`, `fseek` a `fsetpos`.

3.1.3 Použité funkcie z tejto knižnice

3.1.3.1 fopen

```
FILE * fopen (const char * filename,
              const char * mode)
```

Otvára súbor a spája ho so streamom, ktorý identifikujeme v nasledujúcich operáciach ako ukazovateľ na objekt `FILE`, ktorý táto funkcia vracia.

Parametre:

- `const char * filename` - C string, ktorý obsahuje názov súboru na otvorenie.
- `const char * mode` - C string obsahujúci mód prísupu do súboru, typy módov zadávame na základe tabuľky 3.1

Tabuľka 3.1: C stringy pre mód pri otváraní súboru

“r”	Čítanie, orvára súbor na vstupné operácie. Daný súbor musí existovať.
“w”	Zápis, vytvára prázdny súbor pre výstupné operácie. Ak súbor s rovnakým názvom už existuje, jeho obsah je zahodený a so súborom sa zaobchádza ako s novým prázdnym súborom.
“a”	Pripojenie, otvára súbor pre výstupné operácie, pozícia je nastavená na koniec súboru. Výstupné operácie vždy zapisujú dáta na koniec súboru, zväčšujú ho. Operácie na zmenu pozície v súbore sú ignorované. Ak súbor neexistuje, je vytvorený.
“r+”	Čítanie a update. Otvára súbor na aktualizáciu, pre vstupné aj výstupné operácie. Súbor musí existovať.
“w+”	Zápis a update. Vytvára prázdny súbor a otvára ho na aktualizáciu, opäť pre vstupné aj výstupné operácie. Ak existuje súbor s rovankým názvom, jeho obsah je vymazaný a pracuje sa s ním akos novým, prázdnym súborom.
“a”	Pripojenie a update. Otvára súbor na aktualizáciu, pre vstupné a výstupné operácie. Všetky výstupné operácie zapisujú dáta na koniec súboru. Operácie na zmenu pozície v súbore ovlyvňujú nasledujúcu operáciu na vstupe, ale výstupné operácie presunú pozíciu späť na koniec súboru. Ak súbor neexistuje, je vytvorený.

Pri použití vyššie uvedených skratiek jednotlivých módov je súbor otvorený ako textový súbor. Ak chceme otvoriť súbor ako binárny, musí text skratky obsahovať znak „b“. Tento znak „b“ pridávame na koniec stringu,

napríklad „rb“, „wb“, „ab“, „r+b“, „w+b“, „a+b“ alebo ho vložíme medzi písmeno a znak +, teda „rb+“, „wb+“, „ab+“.

V novom štandarde `C2011` bola pridaná nová možnosť „x“. To môže nasledovať za akýmkoľvek špecifikátorom „w“. Táto možnosť prinúti funkciu zlyhať ak už daný súbor s ktorým chceme pracovať existuje, takže existujúci súbor sa neprepíše.

Návratová hodnota:

Pri úspešnom otvorení súboru vracia funkcia ukazovateľ na objekt `FILE`, ktorý môžeme použiť na identifikáciu streamu pri ďalších operáciach. V iných prípadoch vráti funkcia ukazovateľ null. Pri neúspešnom pokuse o otvorenie súboru je premenná `errno` nastavená na chybovú hodnotu špecifickú pre systém na ktorom pracujeme.

3.1.3.2 fclose

```
int fclose (FILE * stream)
```

Zatvára súbor spojený s daným streamom a rozpája ho od streamu. Obsah zo všetkých nevypísaných bufferov spojených so streamom sú vypísané.

Parametre:

- `FILE * stream` - Ukazovateľ na objekt typu `FILE`, ktorý špecifikuje stream na zavretie.

Návratová hodnota:

Ak je stream úspešne zavretý vracia nulovú hodnotu. Pri neúspechu vracia `EOF`.

3.1.3.3 fwrite

```
size_t fwrite (const void * ptr,
               size_t size,
               size_t count,
               FILE * stream)
```

Zapisuje pole veľkosti danej parametrom `count`, každý prvok v tomto poli má veľkosť danú parametrom `size`. Pole zapisuje z bloku pamäte na ktorý ukazuje ukazovateľ `ptr` na momentálnu pozíciu v streame. Indikátor pozície v streame je posunutý o počet zapísaných bajtov.

Parametre:

- `const void * ptr` - Ukazovateľ na pole prvkov, ktoré chceme zapísať, konvertované na `const void*`

- `size` - Veľkosť každého vypisovaného prvku v bajtoch.
- `count` - Počet vypisovaných prvkov, každý prvok je veľkosti danej parametrom `size`.
- `FILE * stream` - Ukazovateľ na objekt `FILE`, ktorý špecifikuje výstupný stream.

Návratová hodnota:

Funkcia vracia počet úspešne zapísaných prvkov. Ak sa toto číslo líši od čísla zadaného parametrom `count` nastala chyba pri zapisovaní. V tomto prípade je v streame nastavený indikátor chyby.

3.1.3.4 fread

```
size_t fread (void * ptr,
              size_t size,
              size_t count,
              FILE * stream)
```

Načítava pole s počtom prvkov daným parametrom count, každý prvok je veľkosti danej parametrom size. Načítava zo streamu a ukladá ich do bloku pamäte na ktorý ukazuje parameter ptr. Indikátor pozície v streame je zvýšený o počet úspešne načítaných bajtov. Pri úspešnom čítaní je celkový počet načítaných bajtov rovný počtu prvkov vynásobeného veľkosťou prvku.

Parametre:

- `void * ptr` - Ukazovateľ na blok v pamäti, ktorého veľkosť je minimálne `size*count` bajtov.
- `size_t size` - Veľkosť každého prvku v bajtoch.
- `size_t count` - Počet prvkov, každý prvok je veľkosti `size`.
- `FILE * stream` - Ukazovateľ na objekt `FILE`, ktorý špecifikuje vstupný stream.

Návratová hodnota:

Funkcia vracia počet úspešne načítaných prvkov. Ak sa tento údaj líši od toho zadaného parametrom `count`, nastala chyba pri čítaní alebo sme dosiali koniec súboru. V oboch prípadoch je nastavený príslušný indikátor, ktorý môže byť kontrolovaný funkciami `ferror` a `feof`.

3.1.3.5 fseek

```
int fseek (FILE * stream,
           long int offset,
           int origin)
```

Nastavuje indikátor pozície v streame na novú pozíciu. Pre streamy otvorené v binárnom móde je nová pozícia definovaná pridánim hodnoty z parametru `offset` k referenčnej pozícii špecifikovanej parametrom `origin`.

Parametre:

- `FILE * stream` - Ukazovateľ na objekt `FILE`, ktorý špecifikuje stream.
- `long int offset` - Pre binárne súbory je to počet bajtov od referenčenj pozície `offset`. Pre textové súbory je to buď 0, alebo hodnota ktrú nám vráti funkcia `ftell`.
- `int origin` - Pozícia použitá ako referenčná. Je zadávaná jednou z nasledujúcich definovaných konštant, `SEEK_SET` je začiatok súboru, `SEEK_CUR` značí aktuálnu pozíciu ukazovateľa a `SEEK_END` znamená koniec súboru.

3.1.3.6 feof

```
int feof (FILE * stream)
```

Kontroluje, či je nastavený indikátor konca súboru. Indikátor je nastavený predchádzajúcou operáciou čítania na streame, ktorá sa pokúša čítať na konci alebo za koncom súboru. Indikátor je zmazaný volaním niektorej z funkcií `cleanerr`, `rewind`, `fseek`, `fsetpos` a `freopen`.

Parametre:

- `FILE * stream` - Ukazovateľ na objekt `FILE`, ktorý špecifikuje stream.

Návratová hodnota:
Nenulová hodnota je vrátená v prípade, že indikátor konca súboru je v danom streame nastavený. Inak vracia nulu.

3.1.3.7 ferror

```
int ferror (FILE * stream)
```

Kontroluje stav indikátoru chyby. Tento indikátor je nastavený predošlou operáciou na streame, ktorá vrátila chybu, je zmazaný volaním `clearerr`, `rewind` alebo `freopen`.

Parametre:

- `FILE * stream` - Ukazovateľ na objekt `FILE`, ktorý špecifikuje stream.

Návratová hodnota:
Ak je na danom streame nastavený chybový indikátor, funkcia vracia nenulovú hodnotu. V opačnom prípade vracia nulu.

3.1.3.8 clearerr

```
void clearerr (FILE * stream)
```

Resetuje indikátor chyby a indikátor konca súboru na danom streame.

Parametre:

- `FILE * stream` - Ukazovateľ na objekt `FILE`, ktorý špecifikuje stream.

3.1.3.9 ftell

```
long int ftell (FILE * stream)
```

Zisťuje aktuálnu hodnotu indikátora pozície v streame. Pre binárne súbory je táto hodnota počet bajtov od začiatku súboru.

Parametre:

- `FILE * stream` - Ukazovateľ na objekt `FILE`, ktorý špecifikuje stream.

Návratová hodnota:
Pri úspešnom volaní funkcie je vrátená aktuálna hodnota indikátoru pozície. Pri zlyhaní vracia hodnotu `-1L` a nastaví `errno` na pre systém špecifickú kladnú hodnotu.

3.1.3.10 rewind

```
void rewind (FILE * stream)
```

Nastavuje indikátor pozície v streame na začiatok súboru. Indikátory chyby a konca súboru sú po úspešnom volaní tejto funkcie vymazané.

Parametre:

- `FILE * stream` - Ukazovateľ na objekt `FILE` ktorým identifikujeme stream.

3.2 Knižnica DVDrecovery

Jedným z cieľov tejto práce je vytvorenie knižnice pre programovací jazyk C. Po rozbalení balíku sa nám zdrojový kód a header súbor tejto knižnice rozbalia na potrebné miesto v systéme. Ak chceme v programe používať funkcie tejto knižnice musíme ju v našom programe includovať a to použitím:

```
#include <dvdrecovery.h>
```

Pri kompilácii programu potom musíme linkovať program s našou knižnicou, napríklad:

```
gcc myprogram.c -ldvdrecovery -o myprogram
```

Knižnica obashuje rôzne funkcie, pomocou ktorých dokážeme načítavať dáta z DVD disku, nájsť poškodené sektory na tomto načítanom disku a spojiť dokopy dáta z dvoch poškodených DVD diskov tak, že výsledkom bude ISO image, ktorý obsahuje obnovené dáta z týchto dvoch rovnakých, ale poškodených DVD diskov.

3.2.1 Štruktúra fileData

Štruktúra na ukladanie základných informácií o načítaných obrazoch DVD diskov. Slúži ako parameter pre ďalšie funkcie, ktoré s ňou pracujú.

const char * filePath — Úplná cesta k súboru, v ktorom chceme mať uložené načítané ISO. Túto cestu zadáva užívateľ pri spúštaní programu.

FILE * file — Ukazovateľ na objekt typu `FILE`, ten je definovaný v knižnici `stdio.h`. Používa sa pri viacerých operáciach pri práci so vstupnými a výstupnými súbormi.

long int fileSize — Veľkosť súboru s cestou `filePath`. Veľkosť súboru sa zistuje z DVD ešte pred načítaním celého orazu tohto disku.

int sizeOfBadSectors — Veľkosť dynamického poľa `badSectors`. Dôležité je pamätať si túto veľkosť hlavne kvôli realokovaniu a zväčšovaniu dynamického poľa `badSectors`. Počiatočná hodnota je 1024.

long int * badSectors — Dynamické pole veľkosti `sizeOfBadSectors`. V tomto poli máme uložené všetky poškodené bloky dát, z ktorých sa nám nepodarilo načítať dáta.

int cntOfBadSectors — Počet prvkov v poli `badSectors`, teda počet chybných blokov na danom DVD disku. Keďze v premennej `sizeOfBadSectors` je veľkosť poľa a tá sa nerovná počtu prvkov v tomto poli, používame na uloženie tejto hodnoty ďalšiu premennú.

3.2.2 Konštruktor štruktúry fileData

Ide o funkciu ktorá slúži ako konštruktor štruktúry `fileData`. Použitím tejto funkcie nainicializujeme premenné v štruktúre. Pokúšame sa otvoriť daný súbor v zadanom móde, ak súbor nedokážeme z akéhokoľvek dôvodu otvoriť, funkcia vracia `NULL` a program končí neúspechom. Nakoniec alokujeme miesto pre pole `badSectors`, v ňom sú uložené informácie o poškodených blokoch dát.

```
struct fileData * fileDataCreate(const char * path,
                                 const char * mode)
```

Parametre:

- `const char * path` - Úplná cesta k vytváranému súboru, do ktorého sa skopíruje obraz disku.
- `const char * mode` - Textový tvar módu, v ktorom sa súbor zadaný cestou `path` otvorí. Módy sú definované v štandardnej knižnici na operácie so súbormi a zároveň sa používajú v niektorých funkciách tejto knižnice.

Návratová hodnota:

V prípade, že sa podarí otvoriť súbor funkcia končí úspešne a vracia ukazovateľ na novovytvorenú a inicializovanú štruktúru, tú sme inicializovali práve volaním tejto funkcie. Ak funkcia `fopen` nedokáže súbor otvoriť, vraciame `NULL` a vypíšeme chybovú hlášku. Pri použití konštruktoru v ostatných funkciách vždy kontrolujeme návratovú hodnotu, pretože ak neotvoríme súbor tak nemôžeme pokračovať ďalej v programe.

3.2.3 Deštruktor štruktúry fileData

Funkcia slúži na uvoľneni prostriedkov alokovaných pre štruktúru a volaním funkcie `fclose` zatvára príslušný súbor o ktorom informácie boli uložené v štruktúre. Používame po ukončení práce so súborom.

```
void fileDataDelete(struct fileData * data)
```

Parametre:

- `struct fileData * data` - Ukazovateľ na instanciu existujúcej štruktúry, ktorej alokované prostriedky chceme uvoľniť a zatvoriť súbor s ktorým štruktúra pracovala.

3.2.4 Fronta

Pre potreby našej knižnice sme si implementovali vlastnú frontu. Na implementáciu tejto fronty využívame jednoduchý, jednosmerne zreťazený spojovací zoznam. Fronta je typu `FIFO`, teda prvý dnu, prvý von.

3.2.4.1 Štruktúra queue

Táto štruktúra nám charakterizuje celú frontu. Obsahuje dva ukazovatele na štruktúry `qNode`. Ukazovateľ `first` obshauje adresu prvého prvku vo fronte. Druhý ukazovateľ, `last` odkazuje na posledný prvok v danej fronte. V prípade, že je fronta prázdna sú obidva ukazovatele nastavené na hodnotu `NULL`. Môže nastať aj situácia, kedy oba ukazovatele ukazujú na rovnaký prvok. To znamená, že vo fronte je práve jeden prvok.

3.2.4.2 Štruktúra qNode

Vyjadruje prvok vo fronte. Obsahuje premennú `data` v ktorej máme uloženú hodnotu s ktorou pracujeme a ukazovateľ `next`. Tento ukazovateľ odkazuje na nasledujúci prvok vo fronte. V prípade, že ukazovateľ `next` je nastavený na hodnotu `NULL`, je fronta buď prázdna, alebo sa jedná o posledný prvok vo fronte.

3.2.5 Pridanie prvku do fronty

Funkcia slúži na pridanie prvku na koniec fronty. Na začiatku si alokujeme miesto v pamäti pre nový prvok. Keďže prvok pridáavame na koniec fronty, hodnotu `next` nastavíme na `NULL` a hodnotu `data` na hodnotu zadanú parametrom. Následne kontrolujeme, či má ukazovateľ na posledný prvok fronty hodnotu `NULL`. Ak áno, znamená to, že fronta je prázdna. Preto nastavíme ukazovatele `first` a `last` na náš nový prvok a funkciu ukončíme. V prípade, že fronta nie je prázdna, nastavíme, aby ukazovateľ `next` v doteraz poslednom prvku fronty ukazoval na nový prvok. Tento nový prvok taktiež nastavíme za posledný prvok fronty.

```
void push(struct queue * q,
          int newData)
```

Parametre:

- `struct queue * q` - Ukazovatateľ na štruktúru, do ktorej pridávame prvok. Táto štruktúra musí byť inicializovaná použitím funkcie `newQueue`.
- `int newData` - Parameter obsahujúci číselnú hodnotu, ktorú ukladáme do fronty a ďalej s ňou pracujeme.

3.2.6 Odobratie prvku z fronty

Funkcia odoberá a vracia prvok nasledujúci vo fronte. Ide o prvok na začiatku tejto fronty. Ako prvé kontorlujeme, či je fronta prázdna. Ak áno, vraciame `NULL`. V prípade, že fronta nie je prázdna vytvárame nový ukazovateľ na prvok `tmp`, ten nastavíme, aby ukazoval na prvok na začiatku fronty. Potom nastavíme na prvé miesto vo fronte nový prvok, ten na ktorý odkazuje ukazovateľ `next` z doteraz prvého prvku. Nasledujúci if kontroluje, či sme sa práve neodobrali z fronty posledný prvok, ktorý v nej bol. Ak áno nastavíme aj ukazovateľ `last` na `NULL`. Funkcia úspešne dobehla do konca a preto vrátime ukazovateľ `tmp`, ktorý ukazuje na prvok odobrnaý z fronty.

```
struct qNode * pop(struct queue * q)
```

Parametre:

- `struct queue * q` Ukazovateľ na štruktúru, z ktorej chceme vybrať prvok.

Návratová hodnota:

Funkcia vracia ukazovateľ na prvok vybraný z danej fronty. Ak je táto fronta prázda vracia hodnotu `NULL`.

3.2.7 Výpis dát z bufferu

Táto funkcia slúži na vypisovanie dát zo zadaného bufferu do zadaného výstupného súboru, ten musí byť otvorený v móde, ktorý umožňuje zápis do tohto súboru.

Program postupne prechádza buffer od počiatočného prvku, až po prvok daný parametrom `pos`. Jeden prvok v buffery sa rovná jednemu sektoru dát, tieto sektory sa postupne vypisujú volaním funkcie `fwrite` do výstupného súboru, ten je takisto zadaný parametrom.

```
void writeOutBuffer(char ** buffer,
                    FILE *outFile,
                    int pos)
```

Parametre:

- `char ** buffer` - Ukazovateľ na buffer. Buffer je dvojrozmerné pole štandardne nastavené na veľkosť 10 000 x 2 048. Číslo 2 048 je veľkosť presne jedného sektora na DVD disku, čiže do bufferu vieme uložiť 10 000 sektorov. Maximálny počet sektorov uložených v buffery môže užívateľ meniť podľa svojich požiadaviek.
- `FILE * outFile` - Ukazovateľ na štruktúru FILE, teda v podstate sa jedná o ukazovateľ na náš výstupný súbor. Do tohto súboru vypisujeme dáta z bufferu.

- `int pos` - Počet prvkov(sektorov), ktoré chceme zo zadaného bufferu vypísať.

3.2.8 Určenie posledného bloku

Po chybe pri čítaní dát z DVD disku, preskakujeme určitý blok. V tejto funkcii používame funkciu `fseek` ktorá slúži na preskočenie daného počtu bajtov v súbore. Ak nám táto funkcia vráti chybu, teda nemôže vykonať skok o daný počet bajtov. To znamená že po preskočení bloku sa dostaneme za posledný bajt načítavaných dát. Potrebujeme zistiť veľkosť bloku dát od našej aktuálnej pozície v načítavanom súbore až po posledný bajt načítavaných dát. Na zistenie týchto informácií slúži práve funkcia `lastBlock`.

```
bool lastBlock(FILE * in,
               long int ** sectorNum,
               int * posInSectorNum,
               int * sizeOfBlock,
               long int * size,
               int blockSize)
```

Parametre:

- `FILE * in` - Ukazovateľ na vstupný súbor z ktorého načítavame dáta a hľadáme poškodené sektory. Týmto vstupom je DVD disk.
- `long int ** sectorNum` - Ukazovateľ na dvojrozmerné pole v ktorom máme uložené informácie o chybných blokoch dát.
- `int * posInSectorNum` - Ukazovateľ na premennú v ktorej je uložená aktuálna pozícia v poli sectorNum. Táto pozícia ukazuje na voľný prvok v poli nasledujúci po poslednom vloženom prvku.
- `int * sizeOfBlock` - Ukazovateľ na premennú v ktorej máme uloženú veľkosť aktuálne prekakovaného bloku dát.
- `long int * size` - Ukazovateľ na premennú ktorá obsahuje veľkosť načítavaného súboru na ktorý odkazujeme v premennej in.
- `int blockSize` - Veľkosť preskakovaného bloku pri načítaní chybného bajtu. Štandardne nastavená na 1 024 000 bajtov.

3.2.9 Načítanie dát z optického disku

Ide o jednu z hlavných funkcii knižnice DVDrecovery. Úlohou tejto knižnice je načítať a nakopírovať obsah optického disku na pevný disk aj napriek poškodeným sektorom na optickom disku. Informácie o poškodených sektoroch si ukladáme pre ďalšiu prácu s týmito načítanými dátami a sektormi.

```
int loadISO(FILE * in,
            struct fileData * data,
            int blockSize,
            int bufferSize)
```

Parametre:

- `FILE * in` - Ukazovateľ na objekt typu `FILE`, ktorým identifikujeme stream. V tomto prípade je tento stream spojený so vstupným súborom, ktorým je optická mechanika.
- `struct fileData * data` - Ukazovateľ na štruktúru `fileData`. Tá obsahuje informácie o súbore, do ktorého ukladáme dáta načítavané z optického disku.
- `int blockSize` - Veľkosť bloku dát, ktorý preskakujeme pri načítaní chybného bajtu. Štandardne nastavená na hodnotu 1 024 000 bajtov.
- `int bufferSize` - Veľkosť bufferu. Počet sektorov v buffery, pričom má každý sektor veľkosť 2 048 bajtov.

Návratová hodnota:
Funkcia vracia hodnotu -2 v prípade, že máme načítané všetky dáta z optického disku a nenašli sme ani jeden chybný sektor. Dáta su teda plne funkčné. Ak sme úspešne dokončili načítavanie, ale našli sme aspoň jeden chybný sektor, vracia funkcia hodnotu 0.

3.2.9.1 Buffer

Funkcia začína alokovaním pamäte na dvojrozmerné pole veľkosti 10 000 prvkov, každý prvok má veľkosť 2048 bajtov. Na alokovanie používame funkciu `malloc`, ktorá slúži na alokovanie bloku danej veľkosti v pamäti a vracia ukazovateľ na začiatok tohto bloku. Keďže potrebujeme alokovať pamäť pre dvojrozmerné pole, pri prvom použití funkcie `malloc` zadáme počet prvkov. Ďalej používame túto funkciu v cykle a alokujeme pamäť pre každý prvok. Toto dynamicky alokované dvojrozmerné pole slúži ako buffer pri načítavaní a vypisovaní dát.

3.2.9.2 Premenné

Hneď po alokovaní si nadefinujeme používané premenné. Premenná `read` slúži na ukladanie hodnoty vrátenej funkciou `fread`, ktorá znamená počet úspešne načítaných bajtov. V ďalšej premennej `position` máme vždy uloženú aktuálnu pozíciu v poli `buffer`. `sizeOfBlock` obsahuje veľkosť aktuálne preskakovaného bloku pri chybnom čítaní. Nakoniec premenná `last` obsahuje Booleovskú hodnotu vrátenú funkciou `lastBlock`.

3.2.9.3 Cyklus while

Dostávame sa do hlavného cyklu funkcie `loadISO`. Tento while volá pri každom svojom opakovaní funkciu `feof`, ktorej zadávame ako parameter náš vstupný stream. Cyklus sa vykonáva, až kým nám funkcia `feof` vráti nenulovú hodnotu. To znamená, že sme na konci načítavaných dát.

Prvý `if` kontroluje hodnotu v premennej `position`. Podmienka je splnená v prípade, že aktuálna pozícia je na konci bufferu. Plný buffer znamená vypísanie dát v ňom uložených do výstupného streamu, preto voláme funkciu `writeOutBuffer` a nastavujeme poz9ciu na hodnotu 0, aby sa načítavané díta zapisovali znova od začiaku bufferu.

Druhý `if` kontroluje zasa premennú `last`, ktorá nám dáva vedieť či sme po preskočení bloku dát na konci načítavaného súboru. Hodnota v premennej `last` je na začiatku nastavená na hodnotu `false`, to znamená, že sa nenachádzame na konci súboru. Naopak hodnota `true` značí, že sme sa preskočením bloku dostali na koniec súboru. V takom prípade využívame príkaz `break`, ten slúži na ukončenie a vyskočenie z cyklu.

Ako ďalšie prebieha načítanie volaním funkcie `fread`. Táto sa vďaka parametrom 1 a 1 snaží načítať len jeden bajt zo súboru. Jeden bajt načítavame hlavne, aby sme vedeli preskočiť blok dát ak sa tento bajt nepodarí načítať. Pri poškodenom optickom disku ide takmer stále o viacero sektorov nasledujúcich hneď po sebe. Optická mechanika sa vtedy spomalí a opakovane sa snaží o načítanie týchto poškodených častí, to značne spomaľuje proces načítavania optického disku, ten sa pri načítavaní po sektoroch aj v prípade chyby nepodarilo dokončiť v rozumnom a pre užívateľa prípustnom čase. Preto na začiatku načítavame tento jeden bajt.

Nasledujúci if kontorluje chybový identifikátor na vstupnom streame pomocou funkcie `ferror`. Tým kontrolujeme ako dopadlo predošlé volanie funkcie `fread`, teda či sa nám podarilo načítať jeden bajt zo súboru.

Bajt sa nepodarilo načítať

V tomto prípade začíname uložením si aktuálnej pozície v načítavanom súbore. Pozíciu ukladáme do poľa `badSectors` v štruktúre `data` a získame ju volaním funkcie `ftell`. V tomto poli máme uložené začiatočné pozície všetkých blokov z ktorých sa nám nepodarilo načítať prvý bajt. Informácie o chybných blokoch využívame v ďalších funkciách pri spájaní dvoch obrazov diskov a obnove dát z týchto diskov.

Ďalej kontrolujeme splnenie podmienky, ktorá má za úlohu zistiť či sme na konci bloku alokovaného pre pole `badSectors`. Ak je podmienka splnená a my sa nachádzame na konci tohto bloku, zdvojnásobíme veľkosť alokovaného bloku. Najprv si v štruktúre `data` vynásobíme dvomi informáciu o aktuálnej veľkosti poľa, tú máme uloženú v premennej `sizeOfBadSectors`. Potom volaním funkcie `realloc` nastavíme veľkosť alokovaného bloku v pamäti na novú hodnotu. Funkcia `realloc` slúži na zmenu veľkosti bloku alokovanej pamäte.

Ak sa daný blok nemôže zväčšiť, existujúci blok sa skopíruje na miesto kde je možné alokovať blok pamäte novej veľkosti.

Voláme funkciu `lastBlock` a hodnotu, ktorú táto funkcia vracia si ukladáme do premennej `last`. Týmto sme zistili či sa po preskočení bloku nachádzame na konci súboru a do premennej `sizeOfBlock` sa nám uloží veľkosť preskočeného bloku. Ak sa nejedná a poslený blok tak veľkosť je 1 024 000 bajtov.

Nasleduje for cyklus. Ten sa vykoná toľko krát, aký je počet sektorov v preskočenom bloku dát. Funkciou `memset` nastavíme v buffery jeden sektor, pričom každý bajt tohto sektoru obsahuje hodnotu z ASCII tabuľky danú ako 0 v desiatkovej sústave. Po zapísaní sektora do bufferu kontrolujeme, či sme sa nedostali na koniec bufferu. Ak sme na jeho konci, vypisujeme dáta do výstupného súboru, o čo sa stará funkcia `writeOutBuffer`.

Na koniec ešte musíme resetovať iniciátor chyby na streame, to vykonávame funkciou `clearerr`.

Bajt sa podarilo načítať

Bajt je úspešne načítaný, potrebujeme teda načítať sektor začínajúci týmto bajtom do bufferu. Ako prvé vrátime použitím funkcie `fseek` pozíciu vo vstupnom streame o jeden bajt späť. Toto musíme vykonať, aby sme mohli načítať celý sektor vrátane tohto prvého bajtu.

Hneď potom načítame do bufferu celý sektor začínajúci daným bajtom. Opäť však môže nastať situácia, že nám funkcia `fread` nenačíta celý sektor úspešne. Vtedy sa nastaví indikátor chyby na vstupnom streame.

Nasleduje `if`, ktorý kontroluje túto chybu pri čítaní, znova funkciou `ferror`. Ak nastala chyba pri čítaní začíname vykonávať príkazy v tomto ife. Najskôr používame funkciu `fseek`, pomocou nej sa v súbore vrátime o počet úspešne načítaných bajtov posleným volaním funkcie `fread`, ktorý máme uložený v premennej `read`. Znova nastáva proces pri ktorom si uložíme začiatočnú pozíciu chybného bloku do poľa `badSectors` v štruktúre `data`. Túto pozíciu zisťujeme funkciou `ftell`.

Voláme funkciu `lastBlock` na preskočenie bloku dát a zistenie, či sme po tomto skoku na konci načítavaného súboru. Návratovú hodnotu funkcie si ukladáme do premennej `last`. Takisto je dôležité, že veľkosť preskočeného bloku máme uloženú v premennej `sizeOfBlock`.

Nasleduje for cyklus, ten sa opakuje podľa počtu sektorov v preskočenom bloku. Najprv nastavíme sektor v buffery na hodnotu z ASCII danú ako 0 v desiatkovej sústave. Na to využívame funkciu `memset`. Tak ako po každom pridaní sektoru do bufferu, kontrolujeme či sa nenachádzame na konci tohto bufferu. Ak sme na konci tak buffer vypisujeme do výstupného súboru volaním funkcie `writeOutBuffer` a nastavujeme pozíciu na 0.

Teraz resetujeme chybový indikátor na streame. To docielime funkciou `clearerr`.

3.2.9.4 Ukončenie načítavania

Po skončení tohto načítavacieho cyklu ešte musíme poslednýkrát vypísať obsah bufferu do výstupného súboru, čiže opäť voláme funkciu `writeOutBuffer`.

Keďže pole, ktoré nám služilo ako buffer už nepotrebujeme, je nutné uvoľniť pamäť, ktorú sme si na začiatku pre toto dvojrozmerné pole alokovali. Najprv prechádzame v cykle postupne každý prvok a uvoľnujeme pamäť ktorá bola pre tento prvok alokovaná, veľkosť prvku je veľkosť jedného sektoru, tj. 2 048 bajtov. Potom uvoľníme zvyšnú alokovanú pamäť. Na uvoľňovanie dynamicky alokovanej pamäte používame funkciu `free`.

Ďalej skontrolujeme v našej štruktúre premennú `cntOfBadSectors`, v ktorej máme uložený počet chybých blokov na optickom disku. Ak je táto hodnota nulová, znamená to, že sme úspešne načítali všetky dáta z optického disku a nie je potrebné načítavať ďalší disk. Informáciu o tomto výsledku vypíšeme užívateľovi a ukončíme funkciu.

Nakoniec vypisujeme obsah poľa `badSectors` čiže vypíšeme začiatocné pozície všetkých chybných blokov, ktoré sme preskočili. Taktiež vypíšeme celkový počet chybných blokov.

3.2.10 Načítavanie po sektoroch z daného bloku dát

Táto funkcia sa využíva v prípade, že preskočené bloky na dvoch optických diskoch sa prekrývajú. Keďže v preskočených blokoch nemusia byt všetky sektory chybné a teda nečitateľné. Potrebujeme zistiť, či sú chybné rovnaké sektory na oboch diskoch. Ak sú chybné zhodné sektory, nemôžeme obnoviť dáta z týchto diskov.

```
int loadBySectors(long int * startPosition
                  long int * endPosition,
                  struct fileData * fileOne,
                  const char * DVDpath)
```

Parametre:

- `long int * startPosition` - Začiatočná pozícia chybného bloku dát v súbore, ktorého informácie sú v parametry `fileOne`.
- `long int * endPosition` - Konečná pozícia bloku dát označeného ako chybný.
- `struct fileData * fileOne` - Ukazovateľ na štruktúru `fileData`. Táto štruktúra obsahuje informácie o súbore do ktorého zapisujeme a v prípade úspechu obsahuje tento súbor všetky obnovené dáta.
- `const char * DVDpath` - Ide o `string`, ten obsahuje cestu k optickej mechanike.

Návratová hodnota:

Funkcia vracia v prípade úspechu hodnotu nula. Ak sa však nepodarí načítať dáta po sektoroch, funkcia vracia nenulovú hodnotu.

3.2.10.1 Premenné

Hneď na začiatku funkcie si nastavíme premennú `actualPosition` na začiatok nami čítaného bloku dát. Táto premenná vyjadruje aktuálnu pozíciu v načítavanom súbore.

Ďalej si vypočítame počet sektorov v bloku, ktorý prechádzame po sektoroch a počet týchto sektorov máme uložený v premennej `numOfSectors`.

Na ukladanie načítaných dát znova používame buffer, ten máme dynamicky alokovaný ako dvojrozmerné pole o veľkosti `numOfSectors` prvkov, pričom každý prvok je jeden sektor, takže má veľkosť 2 048 bajtov. Premennú `positionInBuffer` používame na ukladanie aktuálnej pozície v buffery. Ďalej si vytvoríme frontu `q` a inicializujeme ju volaním funkcie `newQueue`.

Nakoniec máme ešte ukazovateľ na objekt typu `FILE` nazvaný `dvdDrive`. Ten ukazuje na stream ktorý pracuje so vstupným súborom. Tým je v tomto prípade naša optická mechanika.

3.2.10.2 Načítavanie dát

Ešte pred začatím načítavania vypíšeme hlášku, ktorá vyzýva užívateľa na vloženie disku číslo jeden a čakáme kým to užívateľ potvrdí klávesou enter.

Načítavanie prebieha vo while cykle, ktorý sa opakuje, kým sa premennou `actualPosition` nedostaneme na koniec zadaného bloku.

Ako prvé si nastavíme pozíciu na streame, ktorým čítame optický disk. Pomocou funkcie `fseek` si ju nastavíme na `actualPosition`. Pri prvej iterácií je v tejto premennej uložená pozícia ukazujúca na začiatok nami čítaného bloku dát. Pokračujeme čítaním jedného sektora z disku. Úspech funkcie `fread` kontrolujeme použitím funkcie `ferror`.

Ak funkcia `fread` skončila neúspechom, používame funkciu `push` na pridanie prvku do fronty. Vo fronte máme uložené informácie o sektoroch, ktoré sa nepodarí načítať. Funkciou `clearerr` následne zresetujeme indikátor chyby na našom vstupnom streame. Ako posledné v cykle posunieme nassu aktuálnu pozíciu v premennej `actualPosition` o veľkosť jedného sketora.

Po dokončení cyklu zatvárame čitaný stream a vyzývame užívateľa k vloženiu disku číslo dva. Opäť čakáme na potvrdenie klávesou enter a otvárame stream na čítanie pomocou `fopen`. Deklarujeme si premennú, ktorá funguje ako jeden prvok z fronty.

Ďalší cyklus, ktorý sa opakuje, kým nie je prázdna fronta. Funkciou `fseek` zmeníme pozíciu na sektor, ktorý sme vytiahli z fronty. V nej bol, pretože sa nám ho nepodarilo načítať z prvého optického disku. Následne skúšame tento sektor načítať použitím `fread`. Ak skončí fread neúspechom, vieme že tento

sektor nedokážeme načítať ani z jedného disku, preto načítavanie ukončíme, oznamujeme užívateľovi chybu, zatvoríme stream a uvoľňujeme pamäť alokovanú pre buffer.

Ak však cyklus načíta všetky sektory z fronty úspešne, máme plný buffer dát. Najskôr zatvárame stream z ktorého sme načítavali. V súbore do ktorého zapisujeme obnovené dáta si nastavíme pozíciu pomocou `fseek` na začiatočnú pozíciu obnovovaného bloku. Hneď potom do tohto súboru pomocou `writeOutBuffer` vypisujeme celý buffer. Nakoniec uvoľňujeme pamäť alokovanú na začiatku pre buffer.

3.2.11 Zlúčenie dvoch obrazov diskov

Po načítaní obsahu dvoch optických diskov využívame na ich zlúčenie práve túto funkciu. Máme obrazy dvoch diskov a informácie o chybných blokoch dát na každom z nich. Funkcia sa snaží načítať bloky označené ako chybné v prvom obraze z druhého a zapísať ich do prvého. Keďže dáta zapisujeme do prvého obrazu po úspešnom vykonaní tejto funkcie je tento prvý obraz disku plne funkčný a obsahuje všetky obnovené dáta.

```
int mergeImages(struct fileData * fileOne,
                struct fileData * fileTwo,
                const char * dvdDrive,
                int blockSize)
```

Parametre:

- `struct fileData * fileOne` - Ukazovateľ na štruktúru `fileData`, ktorá obsahuje informácie o súbore. Tento súbor je obraz prvého načítavaného optického disku. Chybné sektory obnovujeme a zapisujeme do tohto súboru. V prípade úspechu bude práve tento súbor obsahovať obnovené a funkčné dáta.

- `struct fileData * fileTwo` - Ukazovateľ na štruktúru `fileData`, ktorá obsahuje informácie o súbore. Tento súbor je obraz druhého načítavaného optického disku.

- `const char * dvdDrive` String v ktorom je cesta k optickej mechanike.

- `int blockSize` Veľkosť bloku preskakovaného po načítaní chybného bajtu. Štandardne nastavená na 1 024 000 bajtov.

Návratová hodnota:

Funkcia vracia v prípade úspechu hodnotu 0. Ak niektorá z operácií použitých vo funkcii neprebehne úspešne, funkcia vracia hodnotu -1.

3.2.11.1 Začiatok funkcie a premenné

V štruktúrach `fileOne` a `fileTwo` otvárame súbory, ku ktorým cesta je uložená v premennej `filePath` týchto štruktúr. Súbory otvárame funkciou `fopen` a ukazovateľ na stream spojený s otvoreným súborom ukladáme do premennej `file` v štruktúrach. Ide o súbory, ktoré vznikli načítaním a skopírovaním obrazu optického disku funkciou `loadISO`. Súbor v štruktúre `fileOne` otvárame na čítanie a update, súbor v `fileTwo` len na čítanie.

Ďalej alokujeme miesto v pamäti o veľkosti `BLOCK_SIZE` pre buffer. Premenná `pos` je nastavená na hodnotu 0. V nej máme uloženú aktuálnu pozíciu v poli `badSectors`, v ktorom sú uložené pozície chybných blokov. Premennú `sizeOfBlock` nastavíme na `BLOCK_SIZE`. Táto premenná má informáciu o veľkosti bloku s ktorým aktuálne pracujeme. Táto veľkosť je rozdielna od hodnoty definovanej `BLOCK_SIZE` len v prípade, že sa daný blok nachádza na konci súboru.

3.2.11.2 Cyklus

Na začiatku cyklu kontrolujeme našu aktuálnu pozíciu v poli `badSectors`. Kontrolujeme, či sa nachádzame na konci tohto poľa. Ak sa jedná a posledný prvok tohto poľa, kontorlujeme či má tento posledný blok štandardnú veľkosť bloku, tj. `BLOCK_SIZE`. Hodnotu v premennej `sizeOfBlock` nastavíme buď na spomínanú štandardnú veľkoť bloku alebo vypočítame veľkosť tohto posledného bloku.

Nasledujúcim cyklom prechádzame pole `badSectors` zo štruktúry pre informácie o súbore `fileTwo`. Prvok na aktuálnej pozícii `pos` v poli `badSectors` v súbore `fileOne` testujeme oproti všetkým prvkom v tom istom poli pre súbor `fileTwo`. Ak zistíme že sa ktorékoľvek bloky v týchto dvoch poliach prekrývajú, voláme funkciu `loadBySectors`. Pomocou nej skúšame načítať súbory po sektoroch.

V ďalšom kroku nastavíme pozíciu v súbore `fileTwo` pomocou `fseek`. Pozíciu nastavíme na začiatok sektora, ktorý je chybný v súbore `fileOne`. Následne načítavame dáta zo súboru `fileTwo` do bufferu. Ak čítanie neprebehlo úspešne ukončíme funkciou chybou.

Pozíciu v súbore `fileOne` nastavujeme takiso na začiatok chybného sektoru. Jedná sa o rovnakú pozíciu ktorú sme nastavovali v preošlom kroku v súbore `fileTwo`. Po nastavení pozície vypisujeme buffer do súboru `fileOne`. Týmto výpisom sme opravili blok dát, ktorý sa nám z prvého optického disku nepodarilo načítať.

Po úspešnom dobehnutí celého cyklu máme takto v súbore `fileOne` obnovili chybné bloky dát, do ktorých sme nakopírovali obsah blokov zo súboru `fileTwo`.

3.2.12 Funkcia volaná užívateľom

Užívateľ, ktorý vo svojom programe používa knižnicu `DVDrecovery` volá práve túto funkciu. Tá postupne volá ostatné funkcie obsiahnuté v našej knižnici v určenom poradí s určenými parametrami.

```
int recoverDataFromMirroredDiscs(const char * DVDpath,
                                 const char * ISOonePath,
                                 const char * ISOtwoPath,
                                 int blockSize,
                                 int bufferSize)
```

Parametre:

- `const char * DVDpath` - String obsahujúci cestu k optickej mechanike.
- `const char * ISOonePath` String, ktorým zadávame cestu k súboru, do ktorého načítavame obsah prvého disku. Súbor s týmto názvom nesmie v danej lokalite existovať. Funkcia skončí chybou ak existuje súbor s rovnakým názvom.
- `const char * ISOtwoPath` Týmto stringom zadávame cestu k duhému súboru, do tohto súboru je načítaný obsah druhého optického disku. V zadanej lokalite nesmie existovať súbor s rovnakým názov. Ak taký súbor existuje funkcia skončí chybou a načítavanie sa ukončí.
- `int blockSize` Veľkosť preskakovaného bloku dát pri načítaní chybného bajtu. Štandardne nastavená na 1 024 000 bajtov.
- `int bufferSize` Veľkosť bufferu, ide o počet sektorov v buffery. Každý sektor má veľkosť 2 048 bajtov.

Návratová hodnota:

Funkcia vracia hodnotu 0 v prípade že skončila úspešne. Ak narazí na chybu počas niektorej z funkcií alebo pri operáciách so súbormi vracia hodnotu -1.

3.2.12.1 Priebeh

Začíname otvorením streamu funkciou `fopen`, pomocou ktorého načítavame dáta z optickej mechaniky. Následne používame funkciu `fileDataCreate` na inicializovanie štruktúry `fileData` ktorá obsahuje informácie o obraze prvého optického disku, do ktorého načítavame súbory. Nasledujúcimi príkazmi zisťujeme veľkosť dát načítavaných z optického disku.

Pokračujeme volaním funkcie `loadISO`, ktorá načítava dáta z optického disku. Ak táto funkcia vráti hodnotu -2, znamená to že optický disk nie je poškodený a nie je potrebné pokračovať v načítavaní druhého disku a obnove dát. Preto zatvoríme všetky otvorené súbory a uvoľníme alokovanú pamäť.

V prípade, že sme načítali dáta a disk obsahuje chybné sektory, zatvárame vstupný aj výstupný stream pouzitím `fclose`. Vypíšeme hlášku aby užívateľ vymenil optické disky v mechanike a čakáme na potvrdenie klávesou enter.

V tomto momente podobne ako na začiatku tejto funkcie otvárame vstupný stream spojený s optickou mechanikou, funkciou `fileDataCreate` inicializujeme štruktúru s informáciami o druhom načítavanom disku. Znova zisťujeme veľkosť načítavaných dát na disku.

Voláme funkciu `loadISO` na načítanie dát z druhého optického disku. V prípade úspešného čítania disku bez chybných sektorov postupujeme rovnako ako pri prvom volaní `loadISO`. Pokračujeme zatvorením vstupného a výstupného streamu.

Posledným krokom k obnove dát je volanie funkcie `mergeImages`. Pomocou nej sa snažíme nahradiť chybne načítane bloky dát v súbore `dvdOne` tými istými blokmi úspešne načítanými v obraze disku `dvdTwo`.

Kapitola 4

Testovanie

Na testovanie sme vužili optické disky typu DVD+R a DVD-R. Pre potreby testovania sme vytvorili obraz `ISO` pripravený na vypálenie. Tento obraz disku sme vypálili na dva disky. Pre testovanie sme potrebovali poškodené disky, z ktorých nie je možné načítať niektoré sektory. Poškodenie sme docielili mechanicky, snažili sme sa poškodiť disky na rôznych miestach.

Následne sme spustili náš program na načítanie oboch diskov a ich zlúčenie.

Po tomto kroku sme mali na našom pevnom disku tri obrazy diskov vo formáte ISO.

- Prvý je originálom, ktorý sme vypaľovali na optické disky. Ten používame na overenie, že nami vytvorený obnovený obraz z dvoch diskov je identický.
- Druhý je obraz nahraný z prvého poškodeného disku, jeho časti boli neskôr prepísané časťami z tretieho obrazu. Tento obraz má byť funkčný, identický s prvým obrazom.
- Tretí obraz je nahraný z druhého poškodeného disku. Tento obsahuje chybné bloky, niektoré dáta v tomto obraze sú poškodené.

V poslednom kroku využívame nástroje `diff` a `cmp` a to volaním príkazov:

```
diff obraz_jedna obraz_dva
```

alebo

```
cmp obraz_jedna obraz_dva
```

Tieto príkazy kontrolujú dva súbory bajt po bajte, v prípade že sa v niektorom z bajtov líšia vypíšu, že dané súbory nie su rovnaké. My sme v tomto prípade potrebovali, aby bol prvý obraz disku identický s obrazom druhého disku. To znamená úspešne obnovenie dát z dvoch poškodených diskov použitím našej knižnice a utility.

Tento postup sme opakovali viackrát s rôznymi dátami a rôzne poškodenými optickými diskami.

Záver

Hlavným cieľom tejto práce bol návrh a vytvorenie implementácie knižnice jazyka C, ktorá slúži na obnovu dát z dvoch rovnakých, ale rozdielne poškodených optických diskov a implementácia jednoduchej utility, ktorá pracuje s touto knižnicou. Ďalším cieľom bolo zoznámenie sa a analýza problematiky zálohovania dát na optické média.

V prvej časti práce sme analyzovali zálohovanie dát na optické média. Popísali sme jednotlivé metódy zálohovania a porovnali sme typy optických diskov. Pokračovali sme návrhom implementácie na základe analýzy. Ďalej sme implementovali knižnicu DVDrecovery a utilitu, ktorá s ňou pracuje. Nakoniec sme našu implemetáciu testovali na fyzických médiách.

Obnova dát sa zakladá na princípe, že máme dáta zálohované na dvoch diskoch, jedná sa teda o zrkadlovú zálohu. Pri rôznom poškodení každého z týchto dvoch diskov dokážeme zlúčiť nepoškodené časti a vytvoriť obraz funkčného disku.

Výsledkom tejto bakalárskej práce je funkčná knižnica v jazyku C, pomocou ktorej dokážeme obnoviť dáta z dvoch poškodených optických diskov, na ktorých sú dáta zálohované zrkadlovo.

Literatúra

[1] Rouse, M.: backup [online]. September 2016, [Cited 2018-3-23]. Dostupné z: https://searchdatabackup.techtarget.com/definition/backup

[2] Backup, T. O.: backup [online]. June 2012, [Cited 2018-3-23]. Dostupné z: http://typesofbackup.com

[3] Mohankumar, D.: Compact Disc. How it Works?[online]. . Dostupné z: https://www.electroschematics.com/4997/compact-disc-how-it-works/

[4] Professionals, D. S.: What Are the Different CD and DVD Media Formats Available?[online]. [June] [2017], [Cited 2018-3-30]. Dostupné z: http://www.dell.com/support/article/cz/cs/czdhs1/sln82583/what-are-the-different-cd-and-dvd-media-formats-available-?lang=en

[5] Advameg, I.: DVD Technology[online]. [Cited 2018-4-5]. Dostupné z: http://www.scienceclarified.com/Di-El/DVD-Technology.html

[6] of Encyclopaedia Britannica, T. E.: DVD[online]. [March] [2017], [Cited 2018-4-5]. Dostupné z: https://www.britannica.com/technology/DVD

[7] Mesnik, B.: How Blu-ray Optical Discs Work[online]. [March] [2016], [Cited 2018-4-14]. Dostupné z: https://kintronics.com/how-blu-ray-optical-discs-work/

[8] EaseUS: EaseUS Data Recovery Software[online]. [Cited 2018-4-10]. Dostupné z: https://www.easeus.com/data-recovery-software/

[9] Hoffman, C.: Why Deleted Files Can Be Recovered, and How You Can Prevent It[online]. [September] [2016], [Cited 2018-4-10]. Dostupné z: https://www.howtogeek.com/125521/htg-explains-why-deleted-files-can-be-recovered-and-how-you-can-prevent-it/

[10] cstdio(stdio.h)[online]. [Cited 2018-4-22]. Dostupné z: `http://www.cplusplus.com/reference/cstdio/`

Dodatok **A**

Zoznam použitých skratiek

ASCII American Standard Code for Information Interchange

EOF End of File

FIFO First In, First Out

I/O Input/Output

ISO International Organization for Standardization

USB Universal Serial Bus

WORM Write Once, Read Many

Dodatok B

Užívateľská príručka

Pre používanie našej knižnice DVDrecovery je nutné nakopírovať súbory na správne miesto v systéme. To dosiahneme spustením skriptu `install.sh`. Aby skript prebehol úspešne, musí byť spustený s rootovskými právami. Skript spúšťame napríklad:

```
> sudo ./install.sh
```

Pre použitie knižnice v programe ju includujeme následovne:

```
#include <dvdrecovery.h>
```

Program v jazyku C, v ktorom používame knižnicu DVDrecovery potom kompilujeme pomocou kompilátora GCC:

```
> gcc -Wall main.c -ldvdrecovery
```

Dodatok C

Obsah priloženého CD

- `readme.txt` ..... stručný popis obsahu CD
- `exe` ..... adresár so spustiteľnou formou implementácie
  - `recovery` ..... skompilovaný program utility.c
- `src`
  - `impl` ..... zdrojové kódy implementácie
    - `dvdrecovery.c` ..... zdrojový súbor knižnice DVDrecovery
    - `dvdrecovery.h` ..... hlavičkový súbor knižnice DVDrecovery
    - `install.sh` ..... inštalačný skript
    - `libdvdrecovery.so.1.0` ..... knižnica DVDrecovery
    - `utility.c` ..... zdrojový kód programu
  - `thesis` ..... zdrojová forma práce vo formáte LaTeX
- `text` ..... text práce
  - `thesis.pdf` ..... text práce vo formáte PDF